DESIGN NOTE / REPORT

TITLE USERCODE LANGUAGE

AUTHOR C.A. ASHURST

DATE 17/11/71

COMMENTS BY

CIRCULATION

J.M.G. JONES
D.T. YARDY
C. SIMS
I. COOPER.

CONTENTS

1.0 INTRODUCTION

The Usercode Language for the upwards compatible family of Computers based on the ALP 1, ALP 2 and ALP 3 processors is a symbolic assembler language with defined subsets of the instruction set. The unit of program which is written in the Usercode Language is the Cource or Usercode Module. In general a Usercode program consists of greater than one module and these will be assembled individually and combined by the Integrator. The advantages of such an arrangement include the facility with which a large system may be constructed and the ease with which the standard Usercode Modules may be incorporated into a Usercode program.

The language provides the user with a means of writing instructions in a symbolic form, labeling instructions, forming data, specifying listing formatting, using optional sequences, and implementing a comprehensive Macro System.

MODULES

A program consisting of a number of modules requires them to be combined by Integrator which will perform label linking between them , relocate them to specified relative positions in store and provide data for System Loader and Executive.

A Module is described as being relocatable if it contains no $LOC to an absolute address, it may be considered as having a module float which is to be added to every address, or expression involving an address,which is relative to the start of the module.

ELEMENTS

A Usercode Module is a combination of Usercode Elements, these may be one of the following types:-

Labels
Expressions
Instructions
Directives
Comments

Elements are followed by a terminator. There is no restriction on the character length for any element.

2.0 CHARACTER SET

All source tapes in Usercode Language must be prepared in standard ASCII character code. The following characters only may be used in writing elements of the language.

Upper Case Alphabetic	A - Z
Lower Case Alphabetic	a - z
(Which in the case of the usercode elements are equivalent to corresponding upper case letters)	
Numerals	Ø - 9
Brackets etc.	( ) [ ] ' '
Signs	+ -
Special Symbols	* = # $ % & @ `
Terminators	CR LF ; /
Formatting Characters	space, horizontal tab, comma, form feed, vertical tab.

The remainder of the character code set may occur in user defined character strings provided that they have no special usage on the devices used, eg. EOT on a reader.

2.1. Element Terminators

Usercode Elements, which may be labels, expressions, instructions, directives or comments are delimited by what are termed element terminators. They are used here in a free format language to limit the effect of violations of the syntax.

The element teriminator characters are CR, LF, slash and semi-colon. Where slash occurs the following characters are treated as comment. (see Section 4.0). Any non-null combination of CR, LF, Semicolon and slash, which may only occur as the last character, constitues an element terminator.

2.2. Formatting Characters

Formatting Characters may be used freely in the Usercode Language to improve legibility. They have no syntactic significance and any number of them may occur between or within elements.

Formatting characters will only be treated as normal characters in the case of a user defined character string. eg. %CAR (see Section 5.22).

The formatting characters are space, horizontal tab, comma, form feed, vertical tab.

2.3. Special Characters

* Asterisk

The character * may occur in an expression, in any place where a label would be valid, to represent the value of the current location counter in the object program plus one. In its effect this is a self relative label. (see section 3.3)

= Equals

The character = is used in connection with Macros to separate the parts of a macro call element and to delimit the symbolic parameters in the Macro definition. (see section 5.31)

Number Sign

The character # is used as a marker before a label to indicate that the value of this label is to be made available to other modules. (see section 3.0)

$ % & @ Currency Symbols, Percentages, Ampersand, At

The characters $, %, & and @ are identities of four registers held by the assembler. (see section 5.33)

\ Reverse Slant

The character \ enables the user to prepare free format statements which occupy more than one line on the preparation medium by producing a continuation group. \ causes any following group of the terminators CR LF and semicolon to be ignored by the syntax.

3.0 LABELS

Program parameters and primary store locations may be referenced symbolically by means of labels which may be either satisfied within the module by a declaration, an assignment or in default of these be satified externally at integration time. There is a limit of 126 labels which any one module may require to be satified externally.

It is necessary that all labels within a module may be expressed as the sum of a constant and up to two signed 'floats', where a float is either a label which is to be satified externally or the start address of a relocatable module.

A label is described as absolute if it can be described by a constant alone, and as relocatable if it requires one or two floats.

A label as it occurs in the syntax will consists of between one and six characters. The first character must be alphabetic and any sucessive characters alphnumeric. If a label is preceded by a number symbol ## in either a declaration or a assignment, that label will be made available to other modules at integration time. The ZALA directive (see section 5.36) is equivalent to a ## preceding all label declarations and assignments in a module.

A label may only be assigned/declared once in any module.

3.1. Label Declaration

A label declaration consists of the label followed by a terminator. The label will assume the value of the current location counter which is unchanged by such a declaration. Hence a label may be used to identify an instruction or item of data which it precedes. No label which is declared may commence with the first four characters of an instruction or directive known to the assembler.

3.2. Label Assignment

A label may have a value assigned to it by means of the directives ZASL, ZMLT, and ZDVD (see sections 5.17, 5.18, and 5.19 respectively).
Such a label may either be used to identify an instruction or item of data or alternatively to reference a module parameter. There is no restriction on the first four characters such as in the case with declarations above.

3.3. Self Relative Label *

The use of the character * enables the user to access the current value of the location counter in the object program plus one. This self-relative label may occur in an expression as any label. In the case of an instruction * is the current value that the register S would hold at program execution time.

3.4. Expressions

In the Usercode Language an expression may occur as an Element or part of an Element. To give maximum flexibility to this part of the language, all forms of constants and address expressions have been made inter-changable, and are described by the general term 'Expression'. Expressions whether actually single or double word, are always evaluated with double word precision checked for correct expression width at the end of the evaluation. An expression may have any of the following formats:-

i) Decimal Constant
This is unsigned (implying positive). Leading zero's may be included or omitted as desired.

ii) Signed Decimal Constant
As i) above but preceded by a sign.

iii) Hexadecimal Constant
This is unsigned. It consists of hexadecimal digits preceded by open square brackets and followed by close square brackets. Leading zero's may be included or omitted as desired.
eg. [ØDØA]

iv) Signed Hexadecimal Constant
As iii) above but preceded by a sign.

v) Character Constant
This is unsigned. It consists of ASCII characters read literally and occupying one byte each, without parity. They will be preceded and followed by open single quotes e.g. 'A' or 'C

When more than 4 characters occur between the quote marks the fifth and sucessive charact will be ignored.

vi) Signed Character Constant
As v) on previous page but preceded by a sign.

vii) Compound Expression
This consists of constants of the above types together with label references. The first term may be signed or unsigned, any subsequent terms must be signed. It will be preceded and followed by open and close parenthesis respectively.
e.g. (G26 + 6 - G324)
(-8 + [8ØØØ])
There is no limit on the number of terms to a compound expression.

viii) Signed Compound Expression
As vii) above but preceded by a sign.

The valid range of values for any of the above format depends entirely upon context. Likewise the number of external labels and module floats which may occur in an expression. The following symbols indicate the types of expression which may occur at any given time place in the syntax of the Usercode Language.

<e> Expression, no restrictions

<e,l> Expressions, limited to a maximum of two external labels and modules floats for evaluation.

<e,a> Expression, absolute values only (ie. no external labels), no forward references are allowed.

<e,af> Expression, absolute value only, forward reference allowed.

In order to avoid ambiguity when expressions occur adjacent to each other (e.g. ZDVD, ZPAC) an unsigned decimal constant may not follow any decimal constants.

e.g. + 1624, as two expressions, has no unique conversion , whereas + 16+24 has.

4.0 COMMENTS

A comment commences with a / character, which may terminat any previous element, followed by a user defined string of non-terminating characters and delimitied by a terminato which in this case may not be / .

There is no sytactic requirement about the contents of the user defined string of characters since they will be ignore by the assembler. There is no restriction as to the length of comments, where they may occur or the number that may be written between other elements.

If the first character of the user defined string is a second / character the comment will be listed in the same position as instructions and directives and not in the comment. column.

5.0 DIRECTIVES

Directives are Usercode Elements which cause the Assembler to take some specified action. The possible formats and the action required when each directive is used is described in the following paragraphs.

5.1.	ZSLS	Start Listing
5.2.	ZELS	End Listing
5.3.	ZSOB	Start Object
5.4.	ZEOB	End Object
5.5.	ZOLT	Omit Label Tables
5.6.	ZMOD	Module Name
5.7.	ZSRC	Source Name
5.8.	ZPGE	Move Listing to New Page
5.9.	ZLIN	Move Listing
5.10.	ZLOG	Output Message
5.11.	ZWRN	Output Warning
5.12.	ZLOC	Set Location Counter
5.13.	ZCLR	Clear Stores
5.14.	ZSEQ	Start of Subsequent Module
5.15.	ZEAD	Program Entry Address
5.16.	ZEND	End of Source Module
5.17.	ZASL	Assign Label
5.18.	ZMLT	Multiply and Assign
5.19.	ZDVD	Divide and Assign
5.20.	ZPAC	Form Packed Constant
5.21.	ZPAD	Form Packed Constant (Double Length)
5.22.	ZCAR	Character String Constant
5.23.	ZREP	Repeat Sequence
5.24.	ZDMF	Domain Floated Expression
5.25.	ZDLE	Double Length Expression
5.26.	ZINC	Include Optional Sequence
5.27.	ZOMT	Omit Optional Sequence
5.28.	ZSOP	Start Optional Sequence
5.29.	ZEOP	End Optional Sequence
5.30.	ZCOP	Clear Optional Sequence
5.31.	ZSMD	Start Macro Definition
5.32.	ZEMD	End Macro Definition
5.33.	ZPRG	Load Assembler Register

5.34.	ZTST	Test for Condition True
5.35.	ZMKR	Marker Point
5.36.	ZALA	All Labels Available.

5.1. Start Listing

ZSLS

This directive causes the assembler to list the source module from this point.

5.2. End Listing

ZELS

This directive causes the Assembler to cease listing the source module from this point.

ZSLS and ZELS allow for selective listing of a module and will always be listed themselves. The Assembler will automatically list a module until the first ZELS is found.

5.3. Start Object

ZSOB

This directive causes the Assembler to output the object module from this point.

5.4. End Object

ZEOB

This directive causes the Assembler to cease outputting the object module from this point. by means of ZSOB and ZEOB the object module may either be entirely omitted or partially omitted to allow for example satisfaction of labels.

The Assembler will automatically output the object module until the first ZEOB is found.

5.5. Omit Label Tables

ZOLT

This directive causes the Assembler to omit the output of all label tables.

5.6. Module Name

ZMOD ⟨String⟩

⟨String⟩, the module name consists of from one to six characters. The first character must be alphabetic and any sucessive characters alphanumeric.

This directive defines the name of a module for identification by Integrator. If no name is specified, a default name is used which corresponds to the presence of the directive ZMOD ANON. The module name may independently be used as a label or Macro name.

5.7. Source Name

ZSRC ⟨String⟩

⟨String⟩, is 1-32 characters, any further characters will be ignored.

This directive identifies a Source Module or discrete part of a source module, it provides part of the header of the next page of listing and also puts a comment framed by slash and line feed into the object module to be available for printing by system Loader. Any number of ZSRC directives may occur in a Source Module.

5.8. Move Listing to New Page

ZPGE

This directive causes the Assembler to move to the start of the next page on the listing device. If listing is not to a page orienteddevice it will be ignored.

5.9. Move Listing

ZLIN <e,a f>

This directive to the assembler leaves blank lines on the listing, where <e,af> is the number of blank lines. If such paper motion passes the bottom of a page listing will recommence from the start of the next page.

5.1Φ Output Message

ZLOG <String>

This directive results in the message <String> being output together with the address at which it was found to occur. This output will precede other listings, taking place during the first pass of the assembler.

5.11. Output Warning

ZWRN <String>

This directive is used to list a user defined failure message, <String>, on the Assembler listing. Its occurance will be recorded by a warning counter which is output in the end of assembly report.

5.12. Set Location Counter

ZLOC <e,l>

This directive has the effect of changing the value of the current location count, which refers to the Object Module, to the value given by <e,l>
NOTE: If <e,l> contains a reference to a label which has not previously been declared or assigned in the module it will be noted as an external label. Also if it is declared after a ZLOC which references it a label set twice failure will occur.

5.13 Clear Stores

ZCLR ⟨e⟩

This directive causes Core Loader or System Loader to clear an area of the store from the current location up to but not including the location represented by ⟨e⟩ . The current location counter will remain unchanged.

5.14. Start Of Subsequent Module

ZSEQ

This directive enables the user to define any point in a module from which the next module will start at Integration time. This directive may be omitted, in which case the next module will continue from where the location counter is pointing when the ZEND directive is read.

5.15 Program Entry Address

ZEAD <e, a f>

This describes the Core Loader or Integrator the initial program entry address.
NOTE: When the Integrator is being used, it is more logical to define the entry address by means of the Integration Parameters which describe the program as a whole. It is however made available here for consistency reasons.

5.16 End of Source Module

ZEND

When this directive occurs the Assembler will recognise it as the last element of the source Module, causing the Assembler to start the second pass or if on the second pass to stop assembling.

5.17 Assign Label

ZASL <Label> <e,l>

This directive is used to assign the evaluation of <e,l> to the label. The restrictions applying to ZLOC also apply here.

5.18 Multiply and Assign

ZMUS <Label> <e,a (i)> <e,a (ii)>

This directive performs an arithmetic multiplication. The two expressions <e,a (i)> and <e,a (ii)> are evaluated individually, their product is formed and this value is assigned to the label specified. Any overflow beyond single length will cause an error message to be output by the assembler and the directive will be ignored.

5.19 Divide and Assign

ZDVD ⟨Label (i)⟩⟨e,a (i)⟩⟨e,a (ii)⟩⟨Label (ii)⟩

OR

ZDVD ⟨Label (i)⟩⟨e,a (i)⟩⟨e,a (ii)⟩

This directive performs an arithmetic division. The first expression, ⟨e,a (i)⟩ , is evaluated and forms the dividend. The second expression, ⟨e,a (ii)⟩, is evaluated and forms the divisor The division is performed and the first label has assigned to it the value or the quotient and the second label, when present, has assigned to it the value of the remainder.

5.20 Packed Constant

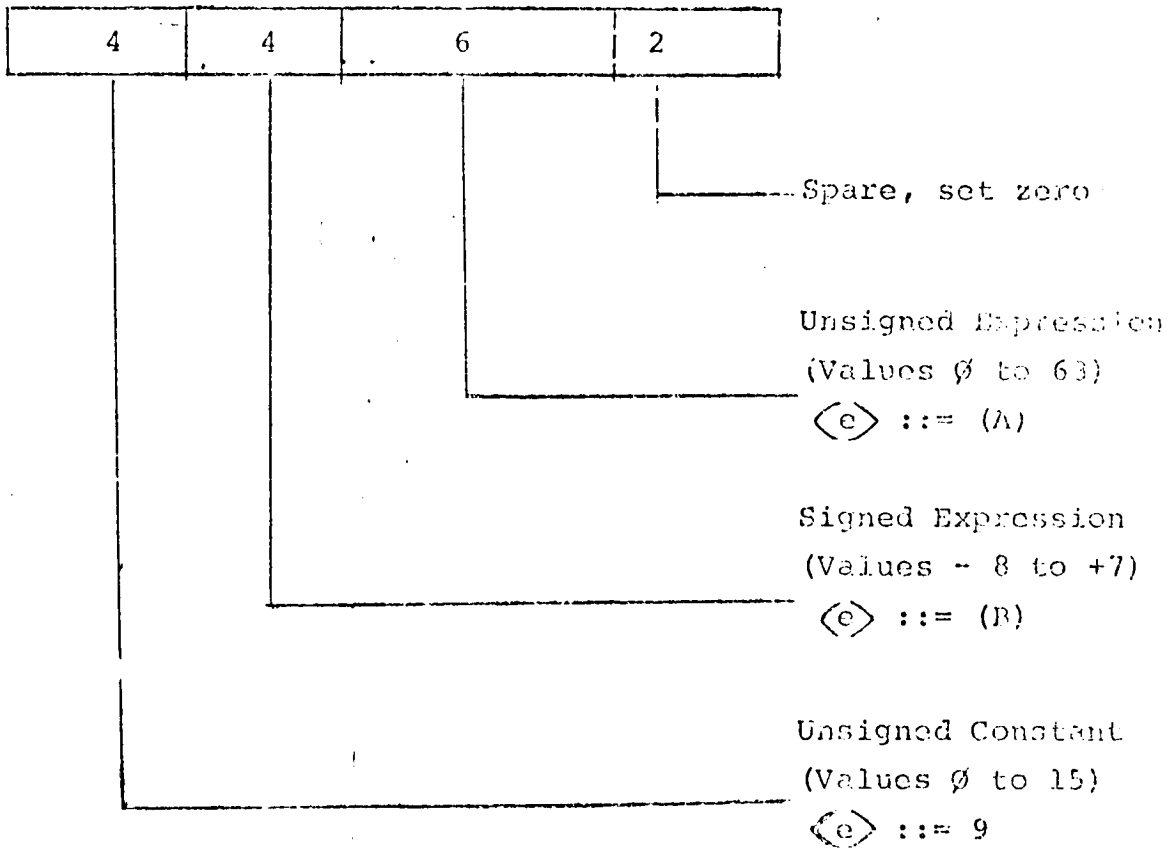
ZPAC ⟨Field (i)⟩ ⟨Field (ii)⟩ .....

Where ⟨Field⟩ ::=S ⟨e,a f⟩ ⟨e⟩ | ⟨e,a f⟩⟨e⟩

This directive enables data to be formed where groups of bits have separate functions. Each field is described by its bit width ⟨e,a f⟩ . its contents ⟨e⟩ and whether it signed, S , or unsigned. Fields will be packed in order from the most significant end of the word towards the least significant end.

If less than 16 bits are specified in total, the unspecified (least significant) bits will be set zero. If more than 16 bits are specified Assembler will fail the ZPAC. Likewise a failure occurs if the field allocated to an expression is overflowed. (This latter failure will be detected by Integrator if external labels or module floats are involved).

EX. PACKED CONSTANT



ZPAC (4) (9) S (4) (B) (6) (A)

5.21 Form Packed Constant (Double Length)

ZPAD has the same form as ZPAC

The operation of this directive is simular to those for ZPAC (5.20).

The one exception is that it produces a double word item. Hence if references to 16 bits are changed to 32 bits in the above description, it can be taken as applying to the ZPAD directive.

NOTE: There are no restrictions about fields which lie across the word boundary.

5.22 Character String Constant

ZCAR ⟨String⟩

This directive specifies to the Assembler a series of character constants to be appear in the object Module. The characters of ⟨String⟩ are read in ASCII code and will be packed, without parity, 2 per word in the Object Module. If there is an odd number of characters in ⟨String⟩, the last word will be zero filled in its least significant byte.

5.23 Repeat Sequence

ZREP ⟨e,a (i)⟩ ⟨e,a (ii)⟩
⟨Element (i)⟩
⟨Element (ii)⟩
...

This directive has the effect of repeating the following series of elements. The number of times of the repeat is ⟨e,a (i)⟩, the number of elements in each repeat is ⟨e,a (ii)⟩

and the elements referred to are ⟨Element (i)⟩ ⟨Element (ii)⟩ etc.

Ex. ZREP (4) (2)
(G298)
+Ø

This directive is equivalent to the sequence of instructions.

(G298)
+Ø
(G298)
+Ø
(G298)
+Ø
(G298)
+Ø

NOTE: A ZREP may have a Macro Call as one of its elements or may occur as an element in a Macro expansion. A ZREP may not have a further ZREP as one of its elements (Either directly or via a Macro Call).

If varying parameters are required, Assembler Registers and ZREG may be used.

5.24 Domain Floated Expression

ZDMF ⟨e⟩

This directive allows programs on ALP2 or ALP3 systems to both refer to locations in a segment which is shared but does not occur in the same store position. It expresses the expression as a displacement from the start of the segment. A typical example of such communications is in an FXEC Call.

When loaded on an ALP1 machine it is simply equivalent to the element ⟨e⟩ alone.

5.25 Double Length Expression

ZDLE ⟨e⟩

This directive causes the expression ⟨e⟩ to be evaluated normally with double word precision and then to be output to the Object Module as a double word constant.

5.26 Include Optional Sequence.

ZINC ⟨e,a⟩

This directive causes optional sequence ⟨e,a⟩ to be included and sequence ⟨e,a⟩ + 100 to be excluded.

NOTE : There are 200 optional sequences arranged as 100 complementary pairs. When reset half of these pairs will be included and half excluded.

OPTIONAL SEQUENCE NUMBER	INITIAL STATE	COMPLEMENTARY TO SEQUENCES
0-49	INCLUDED	100-149
50-99	EXCLUDED	150-199
100-149	EXCLUDED	0-49
150-199	INCLUDED	50-99

Thus for any sequence a charge of 100 in the sequence number will always give a sequence with the opposite condition.
For ZINC and ZOMT only sequences Ø-99 will be specified, since 100-199 will be updated automatically to the other state.

5.27 Omit Optional Sequence

ZOMT <e, a>

This directive causes optional sequence <e, a> to be omitted and <e, a> + 100 to be included.

5.28 Start Optional Sequence

ZSOP <e, a>

This directive delimits the start of an optional sequence. (see ZEOP below)

5.29 End Optional Sequence.

ZEOP <e, a>

This directive delimits the end of an optional sequence. The sequence number <e, a> is in the range Ø to 199.
If such a sequence, delimited by ZSOP and ZEOP, is omitted all elements between them will be ignored except for listing.

5.30 Clean Optional Sequence

ZCOP

This directive causes the optional sequences to be reset to their initial state. Hence the effect of all preceding ZINC and ZOMT directives is cancelled. Optional Sequences Ø-49 and 15Ø-199 will be included and 5Ø-149 will be excluded.

5.31 Start Macro Definition

5.32 End Macro Definition

ZSMD ⟨NAME⟩ ⟨PARAMETERS⟩

⟨CODE⟩

ZEMD

⟨PARAMETERS⟩ ::= =|=⟨SP⟩|=⟨SP⟩⟨PARAMETERS⟩

⟨CODE⟩ ::= ⟨ELEMENT⟩|⟨ELEMENT⟩⟨CODE⟩

⟨NAME⟩ is a four character mnemonic, the first character is alphabetic and the remaining characters alphanumeric. The names ZLOG, ZWRN, ZCAR and ZSRC may not be used.

⟨SP⟩ is a symbolic macro parameter and consists of any string of up to six alphanumeric characters. In any given macro definition there is a maximum limit of 126 symbolic parameters.

⟨ELEMENT⟩ is any usercode element except a macro definition. Elements may contain references to the symbolic parameters in the form =⟨SP⟩=. On expansion of the macro this group will be removed including the delimiting equals characters and replaced by any actual parameters from the macro call. There is no limit to the number of times that each symbolic parameter may be referenced.

For a further details of the macro and examples of its use, refer to section 7.0.

5.33 Load Assembler Register

ZREG ⟨Character⟩⟨e, a⟩

The assembler has four single-word registers which may be used to hold numeric values for such purposes as counters, modifiers and pointers. The registers are represented by single characters which are outside the generally used character set. The characters are $, %, & and @ each identifies a particular register. Thus ⟨character⟩ will consist of one of the above register names.

e.g. ZREG $ +16

These registers are not available inside character constants, comments, ZLOG, ZCAR, ZSRC and ZWRN character strings where they will be treated in the same manner as other characters.

In all other places where the registers are referred to they cause the assembler to read and convert the value currently held by the register. In order to avoid such values being signed, and using the fact that decimal and hexadecimal constants are interchangable, positive values are converted as unsigned, zero suppressed decimal numbers and negative values are converted as hexadecimal numbers delimited by open and close square brackets.

e.g. ␣ ␣ ␣ ␣ 94 (Positive)

[F F F E] (Negative)

Example of the use of ZREG and ZREP

It is required to set up a sequence of two word items for $0 \leqslant n \leqslant 6$

Item n

n	3 - 2n
G19n	

```
ZREG     $   (+ Ø)
ZREP     (7) (2)
ZPAC     (8) ($)          S (8)  (3 - $ - $)
(G19$)
```

This gives rise to the expansion

```
[ØØ 03]
(G19Ø)
[01 05]
(G191)
[02 FF]
(G192)
[03 FD]
(G193)
[04 FB]
(G194)
[05 F9]
(G195)
[06 F7]
(G196)
```

5.34 Test for Condition True.

ZTST ⟨e, a (i)⟩ ⟨Relationship⟩ ⟨e, a (ii)⟩
MKR ⟨e, a (iii)⟩

or

ZTST ⟨e⟩ ⟨Condition⟩MKR ⟨e, a (iii)⟩

⟨Relationship⟩	Interpretation
EQ	Equal to
NE	Not Equal to
GT	Greater than
LE	Less than or Equal to
GE	Greater than or Equal to
LT	Less than

⟨Condition⟩	Interpretation
PD	Previously defined absolutely in this module
ND	Not previously defined in this module

This directive causes the assembler to test the ⟨relationship⟩ between two expressions or the ⟨Condition⟩ of one expression. If the test indicates true, then the next element of usercode to be assembled will be that following the next ZMKR, marker point, with the same numeric operand as ⟨e, a (iii)⟩.

NOTE: All ⟨e, a⟩ terms for this directive are held as double words items. So for example a character constant may consist of up to four characters.

5.35. Marker Point

ZMKR ⟨e, a⟩

This directive is used in conjunction with ZTST directives to indicate the end of a portion of the source module to be omitted following a test finding a true condition. ⟨e, a⟩ is a double word value and will be compared with that specified in the ZTST.
If no search is in progress or the operand ⟨e, a⟩ is not that required the directive is ignored by the assembler, except for listing.

5.36. All labels Available

ZALA

This directive makes the labels declared in this module available to other modules for linking purposes at Integration time. This might conveniently be used with certain types of data module .

6.0 INSTRUCTIONS

Five types of instruction are recognised by the Assembler and these will be referred to as Groups 1, 2, 3, 4 and 5. Some of the instructions will only occur in the subsets of the more comprehensive processors of the range and others require to be run in privileged mode on the machines where this is implemented.

All instructions commence with a four character alphabetic mnemonic which may be followed by an operand definition. In addition to numeric values and expressions the following terms may occur in the instruction.

TERM	REFERENCE
L	Literal Operand
I	Indirect Address
M	Memory Field Register
P	Data Pointer Register
S	Sequence Control Pointer
A	Accumulator Register
B	Accumulator Extension Register
X	Index Register
Y	Index Register
Z	Zero Register
D	All ones Register
ZZ	Zero Condition
PZ	Positive or zero Condition
NN	Negative Condition
PP	Positive Condition
NZ	Negative or Zero Condition
DD	All ones Condition
IZ	Increment and Test Zero Condition
DZ	Decrement and Test Zero Condition

6.1. Group 1 Instructions

These correspond to the Group 1 instructions in the ALP instruction subsets. The mnemonics are as follows:-

MNEMONIC	FORMAT 1 (M=f)	AVAILABLE	INSTRUCTION
STBS		2,3	Store B
STHS		2,3	Store LS Byte A
STES		2,3	Store AB
JUMP	a	u	Jump
LODP	a	u	Load P with Effective Address
STAS	a	u	Store A
LINK	a	u	Link
SETE		2,3	Load AB
ADDE		2,3	Add to AB
SUBE		2,3	Subtract from AB
SETB		2,3	Load B
ADDF		3	Add, Floating Point
SUBF		3	Subtract, Floating Point
MLTF		3	Multiply, Floating Point
DIVF		3	Divide, Floating Point
SETH		2,3	Load LS Byte A
CASH		2,3	Compare LS Byte A
SETA		u	Load A
ADDA		u	Add to A
SUBA		u	Subtract from A
MLTA		u	Multiply with A
DIVE		u	Divide into AB
ANDA		u	AND with A
LORA		u	Inclusive OR with A
XLVA		u	Exclusive OR with A
MASA		u	Mask with A and Skip
CASA		u	Compare with A and Skip
EXAS	a	u	Exchange A with M
INCS	a	u	Increment Memory
DECS	a	u	Decrement Memory

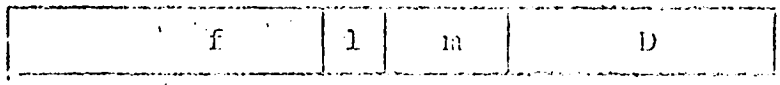
The 'Available' column indicates on which Processor the instruction from part of the subset

u means - available on ALP 1, ALP2, and ALP3
2,3 means - available on ALP2, and ALP3 only.
3 means - available on ALP3 only.

All Group 1 instruction mnemonics must be followed by one the following address formats where it occurs, the expression ⟨e⟩, conforms to the syntax rules in section 3.4.

FORMAT 1

These instuction are held in the memory in the following form:-



The numbers below the diagram indicate the individual bit field widths.

f is the functi..
D is a displacement

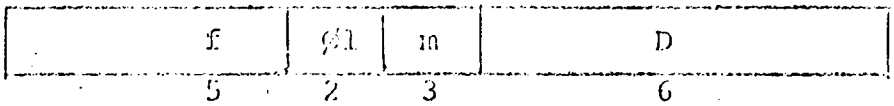
M=Ø

Either	MNEM	L	⟨e⟩
Or for instruction marked 'a'	MNEM	S	⟨e⟩ I
M=1	MNEM	S	⟨e⟩
M=2	MNEM	P	⟨e⟩
M=3	MNEM	P	⟨e⟩ I

MNEM is a four letter mnemonic from the above list
⟨e⟩ is an expression in the range Ø to 4255
⟨e'⟩ is an expression in the range Ø to 4127

FORMAT 2

These instructions are held in the memory in the following form:-



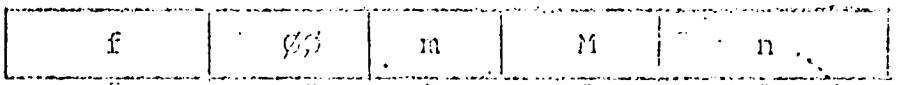
f is the function
m is the number of a Memory Register
D is a displacement.

MNEM M⟨e⟩ ⟨e'⟩

⟨e⟩ has a value in the range Ø to 7
⟨e'⟩ has a value in the range Ø to 63

FORMAT 3

These instructions are held in the memory in the following form:-



f is the function
m is a Memory Register number
n is a Memory Register number

M=Ø	MNEM	M⟨e⟩	M⟨e⟩
M=1	MNEM	M⟨e⟩	M⟨e⟩ I

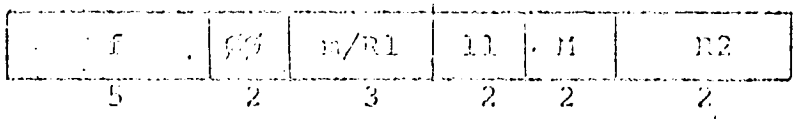
M2 to M5 are only available on ALM2 and ALM3

M=2	MNEM	M⟨e⟩	M⟨e⟩ X
M=3	MNEM	M⟨e⟩	M⟨e⟩ I X
M=4	MNEM	M⟨e⟩	M⟨e⟩ Y
M=5	MNEM	M⟨e⟩	M⟨e⟩ I Y

⟨e⟩ is an expression for the range of values of 7

FORMAT 4

These instructions are only available on ASP 2 and ASP 3. They have the following format:



f is the function

m/R1 is the number of a Memory Register or the code for a Register:-

CODE	SYMBOL	AVAILABLE	REGISTER
0	A	u	Accumulator
1	B	u	Accumulator B [illegible]
2	X	2,3	Index Register
3	Y	2,3	Index Register
4	S	u	Sequence Control
5	P	u	Data Pointer
6	Z	2,3	Zero
7	D	2,3	All Ones

R2 is the code for a Register:-

CODE	SYMBOL	AVAILABLE
0	A	u
1	B	u
2	X	2,3
3	Y	2,3

[illegible]	[illegible]	R(<)	R2
[illegible]	[illegible]	R(<)	R2 +
[illegible]	[illegible]	R1	R2
[illegible]	[illegible]	R1	R2 +

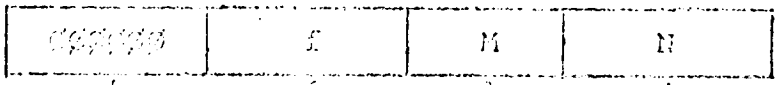
6.2. Group 2 Instructions

These correspond to the Group 2 instructions of the AD' instruction subset. A list of the mnemonics follows:-

MNEMONIC	AVAILABLE	INSTRUCTION
BSOA	u	Set in Bit Register A
BSOB	u	B
BSOX	2,3	X
BSOY	2.3	Y
BSZA	u	Reset bit in Register A
BSZB	u	B
BSZX	2,3	X
BSZY	2,3	Y
SOBA	u	Test Bit in Register A, skip if
SOBB	u	B
SOBX	2,3	X
SOBY	2,3	Y
SZBA	u	Test Bit in Register A, skip if
SZBB	u	B
SZBX	2,3	X
SZBY	2,3	Y
ADMA	u	Add to Register A
ADMB	u	B
ADMX	2,3	X
ADMY	2,3	Y
ADMS	u	S
ADMP	u	P
SBMA	u	Subtract from Register A
SBMB	u	B
SBMX	2,3	X
SBMY	2,3	Y
SBMS	u	S
SBMP	u	P
LSRA	u	Logical Shift Right Register
LSRB	u	Logical Shift Right Register
LSLA	u	Logical Shift Left Register
LSLB	u	Logical Shift Left Register
ASRA	u	Arithmetic Shift Right Register
ASRB	u	Arithmetic Shift Right Register
CSLA	u	Cyclic Shift Left Register
CSLB	u	Cyclic Shift Left Register
LSRE	u	Logical Shift Right Register
LSLE	u	Logical Shift Left Register
ASRE	u	Arithmetic Shift Right Register
CSLE	u	Cyclic Shift Left Register

LDIA	2,3	Load Register A
LDIB	2,3	B
LDIX	2,3	X
LDIY	2,3	Y
LDIS	2,3	S X
LDIP	2,3	P X

The instructions are held in the memory in the following form:-



f is the function

N is a memory register number or a listed constant

M=Ø	MNEM	L	⟨e⟩
M=1	MNEM	M	⟨e⟩

⟨e⟩ is an expression with a value in the range Ø to 15.

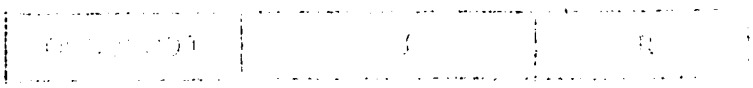
6.3 Group 3 Instructions

These instructions correspond to the Group 3 instructions of the ALP instruction subsets. A list of the mnemonics follows:-

FORMAT 1

MNEMONIC	AVAILABLE	INSTRUCTION
ADRA	u	Add to Register A
ADRB	u	B
ADRX	2,3	X
ADRY	2,3	Y
SBRA	u	Subtract from Register A
SBRB	u	B
SBRX	2,3	X
SBRY	2,3	Y
LDRA	u	Transfer to Register A
LDRB	u	B
LDRX	2,3	X
LDRY	2,3	Y
EXRA	u	Exchange with Register A
EXRB	u	B
EXRX	2,3	X
EXRY	2,3	Y
SERA	u	Compare with Register A, s if Equal
SERB	u	B
SERX	2,3	X
SERY	2,3	Y
SNRA	u	Compare with Register A, s if not Equal
SNRB	u	B
SNRX	2,3	X
SNRY	2,3	Y

The instructions are held in the memory in the following format:-



f is the function

R is the code for a Register as detailed in the following table:-

CODE	REGISTER SYMBOL	
0	A	u
1	B	u
2	X	2,3
3	Y	2,3
4	S	u
5	P	u
6	RESERVED	
7		

MNEM R

FORMAT 2

MNEMONIC	AVAILABLE	INSTRUCTION
STCA	u	Test Condition of Regist A, skip if true
STCB	u	B
STCX	2,3	X
STCY	2,3	Y
SFCA	u	Test Condition of Regist A, skip if false
SFCB	u	B
SFCX	2,3	X
SFCY	2,3	Y

The instructions are held in the memory in the following form.

OPCODE	P	T
8	5	3

f is the function

T is the condition that is to be tested for the possible [illegible]

CODE	CONDITION	AVAILABLE
0	ZZ	u
1	PZ	u
2	MN	u
3	PP	2,3
4	MZ	2,3
5	DD	2,3
6	IZ	2,3
7	DZ	2,3

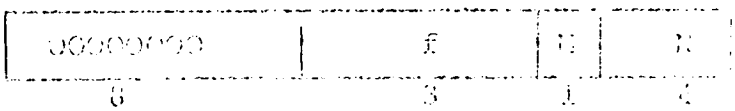
MNEM T

6.4. Group 4 Instructions

These correspond to the Group 4 instructions of the ALP instruction subsets. All of these instruct are privileged. A list of the mnemonics of these instructions follows:-

MNEMONIC	AVAILABLE	INSTRUCTION
INTN	u	Generate Interrupt Sig
RDOM	2,3	Read Domain Register
SDOM	2,3	Set Domain Register
TRGA	u	Trigger A to IOP and
TRGB	u	Trigger B to IOP and
TRGC	u	Trigger C to IOP and
TRGD	u	Trigger D to IOP and

These instructions are held in the memory in the following form:-



f is the function code

N is a [illegible]

L=Ø	MNEM	L	<e>
L=1	MNEM	M	<e>

<e> is an expression which when evaluated has a value in the range Ø to 15.

6.5. Group 5 Instructions

These instructions correspond to the Group 5 instructions of the ALP instruction subsets. Half of them are privileged instructions. A list of the mnemonics follows:-

Privileged Instructions

MNEMONIC	AVAILABLE	INSTRUCTION
HALT	u	STOP Machine
WAIT	u	Idle, Interrupts allowed
INHI	u	Inhibit Interrupts
ALLI	u	Allow Interrupts
REDL	u	Read MIU level to A
SETL	u	Set MIU level from A
RESL	u	Reset MIU level from A
MCTA	u	Machine Conditions to A
ATPC	u	A to Program Condition
ENTL	2,3	Enter Level
SALM	u	Set Alarm
RALM	u	Reset Alarm
SDSU	2,3	Set Domain Unavailable
SDSR	2,3	Set Domain Read only
SDSC	2,3	Set Domain Code only
SDSW	2,3	Set Domain Read/Write

Non-Privileged Instructions

MNEMONIC	AVAILABLE	INSTRUCTION
CLRA	u	Set A to Zero
CLRB	u	Set B to Zero
MNEA	u	Set A to -1
MNEB	u	Set B to -1
SNCO	u	Skip on no Carry
SNAO	u	Skip on no Overflow
EXEC	u	Exective Service Request
TEST	u	Test Point
NEGA	2,3	Negate A
NEGB	2,3	Negate AR
NEGF	3	Negate, Floating Point
FLTI	3	Float Integer
FLTF	3	Float Fraction
FIXI	3	Fix Integer
FIXF	3	Fix Fraction
STND	3	Standardise.

These instructions are held in the memory in the following format

000000000000	f
11	3

f is the fnction

Since there is no operand, the only valid format is simply:-

MNEM

7.0 MACROS

7.1. Introduction

The usercode language provides a comprehensive Macro facility. It allows the user to define a 'macro instruction' as sequences of ordinary usercode elements and provides a means of inserting variable information in the generated sequence.

Consider as an example a subroutine call where the subroutine is to be obeyed with a different position of P than the main program.
A typical entry may have the following form, the underlined parts being the variables.

```
LODP     P (0) I
LINK     P - 1
JUMP     P (2) I
LODP     P (10) I
```

This may be converted to a macro definition as described in section 5.31, using A, B and C as symbolic parameters.

```
ZSMD      SRTN = A = B = C
LODP      P (=A=) I
LINK      P -1
JUMP      P (=B=) I
LODP      P (=C=) I
ZEND
```

Following such a definition the macro may be used by writing a macro call which supplies the actual parameters, thus:-

```
SRTN = LOCN = LOCN + 100
```

When this macro call is encountered the assembler substitutes the macro definition, replacing the symbolic parameters by the actual parameters. So for the call written above, the substitution would be:-

```
        LODP    P (LBTW) I
        LINK    P -1
        JUMP    P (LBTH) I
        LODP    P (LEF) I
```

The normal process of assembly now applies as if the substitued material had actually occured in the program.

A further enhancement of the power of such a tool is to allow macro calls to be nested. That is, it allows any macro definition to contain calls to other macros. It should be noted that when macro calls are nested, actual parameters may be handed from macro call to a macro which it in turn calls.

For example consider the above call which is in a certain application may require a code to be loaded to the accumulator before the subroutine is obeyed and a second code to be loaded after the subroutine has been obeyed.

A macro may be defined which uses the previously defined SRTN macro.

```
ZSMD     PSRN  =  K1 = A = B = C = K2
SETA  L (=K1=)
SRTN  =  =A=   =  =B=   =  =C=
SETA  L (=K2=)
ZEND
```

So, also introducing a default for K2, the above macro may be called:

PSRN = 12 = [illegible] = [illegible] = [illegible]

This [illegible] call would [illegible] be [illegible]

```
SETA    L    (12)
LODP    P    (LBTW) I
LINK    P    -1
JUMP    P    (LBTH) I
LODP    P    (LBF') I
SETA    L    ()
```

Here the value for K2 has been omitted and the expansion gives the element SETA L (), where () is converted by the assembler as +0

7.2 Applications

There are five main areas where it is envisaged that the macro facilities will have most apparent benefit to the user.

1. Instruction Sequence :
This is the type of application which is described in the example in section 7.0. This field of application includes such things as EXEC Calls and entries to standard subroutines or any other recurring feature of uncode language.

2. Data Sequences :
This is typically a list item or common data element which may be one or more words, possibly packed in bit fields. An example is the cell for IOP.

3. New Instruction :
This is a means of providing an instruction set for a new processor or an Interpreter.

4. New Languages :
Problem oriented high-level languages may be implemented in such areas as process control, list processing and mathematics packages.

5. Generation of Standard Packages :
Macros may be used to establish a variable number of variable length queues and to set up variable format items. Options and tests on parameters may take place within the macro definition.

7.3. Examples

1. A Macro to Transfer Control to a given label

Consider the definition:

```
ZSMD    GOTO    = LABEL
ZTST    (=LABEL=) ND  MKR  'A'
ZTSP    (=LABEL= - *) GE ( +128)  MKR 'A'
ZTST    (=LABEL= - *) LE ( -128)  MKR 'A'
JUMP    S (=LABEL= - *)
ZTSS    (Ø) FQ (Ø)   MKR 'B'
ZMKR 'A'
JUMP    S  Ø  I
(=LABEL=)  ,
ZMKR 'B'
ZEMD
```

If the given label has been defined previously in the module and the condition $-128 \leqslant (LABEL - *) < 128$ is true then a single word S relative jump is assembled. Otherwise an S indirect, 2 word, jump is produced. The macro is called by the following statement which generates one of the successive jumps according to the above conditions.

```
GOTO    =  NXTTTT:

Either  JUMP  S  (NXTTTT - *)
  or    JUMP  S  ØI
              (NXTTTT:)
```

2. A Macro which establishes a Queue Area.

The queue is defined as a number of several words items, chained by the first word of each item containing the address of the next item and the final item containing the address ϕ.

Consider the macro definition

```
ZSMD    QUAR    = M = N
ZTST    (=N=) LE (+1)  MKR 'A'
ZREP    (=N= - 1) (2)
(* + = M = - 1)
ZLOC    (ϕ + = m = - 2)
ZMKR    'A'
(ϕ)
ZLOC     (* + = M = - 2)
ZEND
```

M is the number of words per item
N is the number of items

At least one item will be produced. Thus the macro is expanded by a call of the form:

```
QUAR    = 6 = 24
```

3. A Macro to form Transfer Parameter Items:

Each item consists of six words

3	1	3	2	7
F	Ø	M	ØØ	N
DL				
DC				
ST - Ø				
D1				
D2				

Either or both of D1/D2 may be omitted

This may be represented by the macro definition:

```
ZSMD      TRPA  = F = M = N = DL = DC = D1 = D2
ZPAC      (3)  (=F=)   (1) (0)    (3) (=M=)   (2) (0)   (7) [illegible]
ZDMF      (=DL=)
(=DC=)
Ø
ZTST      '=D1='   EQ    (+Ø) MKR 'A'
(=D1=)
ZMKR      'A'
ZTST      '=D2='  EQ   (+Ø) MKR 'B'
(=D2=)
ZMKR      'B'
ZEND
```

This macro will be expanded by the call:

```
TRPA  =Ø= 4 = +14 = BUFFER = +11
```

NOTE That D1 and D2 have been omitted. This [illegible] is equivalent to the sequence of instructions.

```
[ØØ,4]
ZDMF (BUFFER)
(+11)
Ø
```